**262. Trips and Users**

The Trips table holds all taxi trips. Each trip has a unique Id, while Client\_Id and Driver\_Id are both foreign keys to the Users\_Id at the Users table. Status is an ENUM type of (‘completed’, ‘cancelled\_by\_driver’, ‘cancelled\_by\_client’).

+----+-----------+-----------+---------+--------------------+----------+

| Id | Client\_Id | Driver\_Id | City\_Id | Status |Request\_at|

+----+-----------+-----------+---------+--------------------+----------+

| 1 | 1 | 10 | 1 | completed |2013-10-01|

| 2 | 2 | 11 | 1 | cancelled\_by\_driver|2013-10-01|

| 3 | 3 | 12 | 6 | completed |2013-10-01|

| 4 | 4 | 13 | 6 | cancelled\_by\_client|2013-10-01|

| 5 | 1 | 10 | 1 | completed |2013-10-02|

| 6 | 2 | 11 | 6 | completed |2013-10-02|

| 7 | 3 | 12 | 6 | completed |2013-10-02|

| 8 | 2 | 12 | 12 | completed |2013-10-03|

| 9 | 3 | 10 | 12 | completed |2013-10-03|

| 10 | 4 | 13 | 12 | cancelled\_by\_driver|2013-10-03|

+----+-----------+-----------+---------+--------------------+----------+

The Users table holds all users. Each user has a unique Users\_Id, and Role is an ENUM type of (‘client’, ‘driver’, ‘partner’).

+----------+--------+--------+

| Users\_Id | Banned | Role |

+----------+--------+--------+

| 1 | No | client |

| 2 | Yes | client |

| 3 | No | client |

| 4 | No | client |

| 10 | No | driver |

| 11 | No | driver |

| 12 | No | driver |

| 13 | No | driver |

+----------+--------+--------+

Write a SQL query to find the cancellation rate of requests made by unbanned clients between **Oct 1, 2013** and **Oct 3, 2013**. For the above tables, your SQL query should return the following rows with the cancellation rate being rounded to *two* decimal places.

+------------+-------------------+

| Day | Cancellation Rate |

+------------+-------------------+

| 2013-10-01 | 0.33 |

| 2013-10-02 | 0.00 |

| 2013-10-03 | 0.50 |

+------------+-------------------+

首先，我们创建表格Trips 和 Users：code 如下（因为用到表Trips 里的foreign key references了表Users，我们先创建表Users 后创建表Trips ）

create table Users (

Users\_Id tinyint unique,

Banned enum('Yes','No'),

Role enum('client','driver','partner'));

create table Trips (

Id tinyint unique,

Client\_Id tinyint,

Driver\_Id tinyint,

City\_Id tinyint,

Status enum('completed','cancelled\_by\_driver','cancelled\_by\_client'),

Request\_at date,

foreign key(Client\_Id) references Users(Users\_Id),

foreign key(Driver\_Id) references Users(Users\_Id)

);

insert into Users values

(1,'No','client'),

(2,'Yes','client'),

(3,'No','client'),

(4,'No','client'),

(10,'No','driver'),

(11,'No','driver'),

(12,'No','driver'),

(13,'No','driver');

insert into Trips values

(1,1,10,1,'completed','2013-10-01'),

(2,2,11,1,'cancelled\_by\_driver','2013-10-01'),

(3,3,12,6,'completed','2013-10-01'),

(4,4,13,6,'cancelled\_by\_client','2013-10-01'),

(5,1,10,1,'completed','2013-10-02'),

(6,2,11,6,'completed','2013-10-02'),

(7,3,12,6,'completed','2013-10-02'),

(8,2,12,12,'completed','2013-10-03'),

(9,3,10,12,'completed','2013-10-03'),

(10,4,13,12,'cancelled\_by\_driver','2013-10-03');

解题思路：

求的是 cancellation rate of requests made by unbanned clients between **Oct 1, 2013** and **Oct 3, 2013**。

分析：后面的between **Oct 1, 2013** and **Oct 3, 2013** 只是一个限制范围的语句可以用where实现。真正重要的是求每天里（group by 天）的cancellation rate。中间的made by unbanned clients 只是一个外来的限制而已（主要的查询在Trip表，用User表里的字段的值来限制 可以用inner join 来实现，或者书写的时候就写 from Trip，Users where [链接限制条件语句]）。

select \*

from Trip T, Users U

where T.Client\_Id = U.Users\_Id

and U.Banned = ’No’

and T.Request\_at between '2013-10-01' and '2013-10-03'

group by T.Request\_at;

现在来处理每一个group里的cancellation rate：

分母是容易的，用count(\*)即可算出特定group里的行数，即与Banned＝‘No’相关的行数。我们处理分子。分子要求算出cancel 包括了cancelled\_by\_driver and cancelled\_by\_client的数目。这可以用 sum(if(T.Status=’completed’,0,1)) 来计算得到。

*[这里的if(expr1,expr2,expr3) 的语法是如果expr1为真则取expr2的值，否则取expr3的值]*

因此整个的query可以写成：

select T.Request\_at as Day, round(sum(if(T.Status=’completed’,0,1))/sum(\*), 2) as ’Cacellation Rate’

from Trip T, Users U

where T.Client\_Id = U.Users\_Id

and U.Banned = ’No’

and T.Request\_at between '2013-10-01' and '2013-10-03'

group by T.Request\_at;

[*这里round(expr,N) 的语法是返回expr值，四舍五入到小数点后第N位；round(x)的语法是返回离x最近的整数值*]

整个的写MySQL query语句的经验就是，先搞好from后的语句，然后搞好select里的语句。原因如下：from后面总是跟的一个表格，不管是笛卡尔积还是子查询得来的，总是一个从不同的表格间经过处理后得出针对问题的一个基础的表格（基础在这里的意思是一个平台，查询的东西都可以由这个表格里的字段经过MySQL里已有的语言和函数计算得出）。

显然，这样的基础的表格总是存在的。比如，对相关表格做 join（需要的话，同一个表格也可做很多次，只是链接条件不同而已）的肯定能包括所有要用的信息。这里关键在于要使得得到的基础表格“好用”：在写select后的东西的时候好用。

MySQL查询语言的技巧在于怎样

1.写from后表格间的连接条件语句

2.写select后面的选择语句

**197. Rising Temperature**

Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

+---------+------------+------------------+

| Id(INT) | Date(DATE) | Temperature(INT) |

+---------+------------+------------------+

| 1 | 2015-01-01 | 10 |

| 2 | 2015-01-02 | 25 |

| 3 | 2015-01-03 | 20 |

| 4 | 2015-01-04 | 30 |

+---------+------------+------------------+

For example, return the following Ids for the above Weather table:

+----+

| Id |

+----+

| 2 |

| 4 |

+----+

首先创建Weather表：

CREATE TABLE Weather (

Id tinyint NOT NULL,

Date date,

Temperature tinyint,

primary key (Id)

);

insert into weather values

(1,’2015-01-01’,10),

(2,’2015-01-02’,25),

(3,’2015-01-03’,20),

(4,’2015-01-04’,30);

解题思路：

要求all 今天的 Ids 要求今天的 temperature 比昨天的高。我们先写出一个基础的供选择的表格，即select后的内容。要对两天的temperatures比较，那样的话要有两个Weather表格（因为MySQL读表是一条记录一条纪录读的，同一条记录里只能含有一天的信息，这里要含有两天的信息，因此要两个表格） ，把他们inner join起来。如下

select …

from Weather W1, Weather W2

where …

现在写基础表格的限制语句，目的是使得基础表格的一条记录同时含有今天（W1）和昨天（W2）的信息。

to\_days(W1.Date) = to\_days(W2.Date) + 1

[*这里to\_days(date)函数的作用是给定一个日期，返回一从year 0算起的天数。可以用select to\_days(’2015-01-10’)；来试一下*]

然后要求今天的温度比昨天的温度高

W1.Temperature > W2.Temperature

select后面的内容在本题中比较简单： select W1.Id

因此， 完整的query如下：

select W1.Id

from Weather W1,Weather W2

where to\_days(W1.Date) = to\_days(W2.Date)+1

and W1.Temperature > W2.Temperature;

**196. Delete Duplicate Emails**

Write a SQL query to delete all duplicate email entries in a table named Person, keeping only unique emails based on its *smallest* **Id**.

+----+------------------+

| Id | Email |

+----+------------------+

| 1 | john@example.com |

| 2 | bob@example.com |

| 3 | john@example.com |

+----+------------------+

Id is the primary key column for this table.

For example, after running your query, the above Person table should have the following rows:

+----+------------------+

| Id | Email |

+----+------------------+

| 1 | john@example.com |

| 2 | bob@example.com |

+----+------------------+

首先创建表格Person：

因为我的名为cfang\_db的数据库里已经有了Person表格，决定删除重新搞一遍。

drop table if exists Person;

[*这里的语法是：1. drop table [table\_name is ture]; 2. if exists [table\_name] 另外需注意的是MySQL不区分大小写* ]

create table Person (

Id tinyint unique,

Email varchar(30)

);

insert into Person values

(1, 'john@example.com'),

(2, 'bob@example.com'),

(3, 'john@example.com');

解题思路：

题目要求delete all duplicate email entries 只keep email entry相同的纪录中 Id最小的那个。

这里有两个想法，第一个是用inner join，第二个是用子查询。先看第一个

为了删除的时候有针对性，基础表中每一条记录中要有如下几个信息：第一是第一个表的email和第二个表的email要相同；第二是第二个表的Id最好是同一个email中Id小的那个，以便于用第一个表的Id来与之比较，从而确定究竟要不要保留这条纪录。这样的话第二个表就要先预处理一下才能得到。例如(select min(P2.Id) as Id\_min, P2.Email from Person P2 group by P2.Email) tmp可以先运行看看这条语句的结果。现在我们来写from之后的语句

from Person P1

inner join (select min(P2.Id) as Id\_min, P2.Email from Person P2 group by P2.Email) tmp

on tmp.Email = P1.Email

再写基础表格的限制语句。注意这是要删除Id较大的，而不是查询出Id最小的。

where P1.Id>tmp.Id\_min

最后写delete语句（相当于前面的讲的select语句）：

delete P1.\*

from …

合起来就是：

delete P1.\*

from Person P1

inner join (select min(P2.Id) as Id\_min, P2.Email from Person P2 group by P2.Email) tmp

on tmp.Email = P1.Email

where P1.Id>tmp.Id\_min;

现在讲第二个想法，用子查询实现：

我们打算把符合删除的特征写进一个where语句。即where语句里含有相应的email的最小的Id。这是可以实现的：我们的主查询在Person表格里，这里的每一条纪录就已经包含了Email的信息和纪录的Id信息。我们可以用Email信息来限制where后的自查询，然后在自查询里求一个Id的最小值即达到目的。

delete

from Person P1

where P1.Id>(select min(P2.Id)

from Person P2

where P2.Email = P1.Email);

上面的查询语句有问题，我不知道问题出在哪里。下面的可以：

delete

from Person

where Id not in (select \*

from ( select min(Id) from Person group by Email) tmp);

我知道上面的语句的问题出在哪里了^\_^

首先从逻辑上讲，没有问题，把delete改成select \* 还是是有正确的值出现。其次，从MySQL的语法上讲也有问题（可能设计表锁等我目前不熟悉的内容）。改成delete就出问题了！看看第二个查询语句，如果我们把where后的语句改成

where Id not in (select min(Id) from Person group by Email);

这样貌似也不是work的，会出现如下错误：

ERROR 1093 (HY000): You can't specify target table 'person' for update in FROM clause

参考如下网页：

<http://stackoverflow.com/questions/4429319/you-cant-specify-target-table-for-update-in-from-clause>

由于在对表格进行修改（update，delete，alter）时，在from语句里不能出现修改表格的名字，我暂时没有想出fix的办法。这个在inner join第一种方法中已经出现过

<http://stackoverflow.com/questions/45494/mysql-error-1093-cant-specify-target-table-for-update-in-from-clause>

上面的网址也可以参考。

**185. Department Top Three Salaries**

The Employee table holds all employees. Every employee has an Id, and there is also a column for the department Id.

+----+-------+--------+--------------+

| Id | Name | Salary | DepartmentId |

+----+-------+--------+--------------+

| 1 | Joe | 70000 | 1 |

| 2 | Henry | 80000 | 2 |

| 3 | Sam | 60000 | 2 |

| 4 | Max | 90000 | 1 |

| 5 | Janet | 69000 | 1 |

| 6 | Randy | 85000 | 1 |

+----+-------+--------+--------------+

The Department table holds all departments of the company.

+----+----------+

| Id | Name |

+----+----------+

| 1 | IT |

| 2 | Sales |

+----+----------+

Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows.

+------------+----------+--------+

| Department | Employee | Salary |

+------------+----------+--------+

| IT | Max | 90000 |

| IT | Randy | 85000 |

| IT | Joe | 70000 |

| Sales | Henry | 80000 |

| Sales | Sam | 60000 |

+------------+----------+--------+

首先创建表Employee 和 Department

create table Department (

Id tinyint primary key,

Name varchar(30));

create table Employee (

Id tinyint primary key,

Name varchar(20),

Salary int,

DepartmentId tinyint,

foreign key(DepartmentId) references Department(Id));

insert into Department values

(1,’IT’),

(2,’Sales’);

insert into Employee values

(1,’Joe’,70000,1),

(2,’Henry’,80000,2),

(3,’Sam’,60000,2),

(4,’Max’,90000,1),

(5,’Janet’,69000,1),

(6,’Randy’,85000,1);

解题思路：

要求的是找出each department中top three的salary。实现each department 用group by 语句在这里好像不行，因为要求的是每个系的前3，而group by后我只见过用一个函数求出一个值的。所以要考虑逐行扫描判断，当然这不是坏事，MySQL就是一行一行读的。输出的时候对deparment要以相应的名字输出，这只需对两个表格做一个inner join即可。因此，目前的状态是：

select …

from Employee E, Department D

where E.DepartmentId=D.Id

and …

下面我们进一步考虑目前的基础表格是不是够用？从子查询的角度来讲应该是够用了（因为不用修改表格，所以也不会出现上题中的问题）。如果不打算用子查询，则不够。因为select选定基础表格的每一条纪录后，一条记录中并不包含“top three”的信息。我看到过用user-defined variable来解决这个问题的。User-defined variable这个玩意的好处是，对用户定义的变量值可以变。比如在执行上一条纪录的选择时，根据的程序我们赋予这个变量一个新值，当MySQL执行检查下一条纪录是否符合条件时所用到的用户变量的值将是新的值。这个好处是显而易见的：比如我们可以设定一个用户变量，当找到一条合服规定的记录时，就让这个值加一，否则不变，直到这个变量到达3为止。具体见下面的解答。

方法一，使用子查询实现。

基础表格就是上面的inner join之后的表，我们处理where后面的部分。假设MySQL在读基础表格的一条记录了，那么where后面的限制要达到的效果是：只要本记录中的salary 在前3就选出。剩下的是要实现前3，这里涉及子查询。子查询的前提是我们已经直到了这条记录里的salary和departmentid，子查询实现的是这是不是前3。因此我们可以做这样的一个比较，如果把新表（或许就是employee的另一个copy）中比这个salary值大的 新表.salary 的数目算出来，只要这个值取0，1，2或者是小于3即可。写出来就是：

(select count(distinct(E2.Salary))

from Employee E2

where E2.Salaary > E.Salary

and E2.DepartmentId = E.DepartmentId) in (0,1,2)

最后处理select里的内容，这里比较容易:

select D.name as Department, E.Name as Employee, E.Salary

再来调整输出的顺序

order by E.DepartmentId, E.Salary desc;

[*在排序的时候尽量用数字变量排，速度会快些*]

合起来就是

select D.Name as Department, E.Name as Employee, E.Salary

from Employee E, Department D

where (select count(distinct(E2.Salary))

from Employee E2

where E2.DepartmentId = E.DepartmentId) < 3

and

E.DepartmentId = D.Id

order by E.DepartmentId, E.Salary desc

如果是Top N的话创建函数，然后把3用N代替

方法二，用user-defined variable:

这里的想法是还是只对Employee表和Department表做一次查询，但是增加几个用户定义的变量，来存储上次查询或设定的值。这样的话要先对join的表先order by department, salary进行排序，从而使得只需要再做一次查询。我们大概要用到下面几个用户定义的变量，关键的一个是用来计算当前salary的排名，排名初值从1开始，如果在查询下一条记录时发现在同一个系里而且salary减少了则增1，如果在同一个系里而且salary保持不变则排名不变，如果发现系不同了则要把排名归到1，因为这是要算另外一个系的前3名了。最后在排名大于3时就都不要了，这个可以用where语句实现。通过分析，我们发现还需要另外两个用户定义的变量，一个用来存储前一条记录中的系名或者是系Id，一个用来存储前一条纪录中的salary用于和当前纪录比较。

我们使用同样的基础表格 from Employee E, Department D,但是用user-defined variables来协助查询

…

select …

from (select \*

from Employee E, Department D

where E.DepartmentId = D.Id

order by E.DepartmentId, E.Salary desc) tmp1,

(select @Department\_pre:=null, @Salary\_pre:=0, @count:=1 from dual) tmp2

~~where~~ ＃这里是一个平凡的join不需要where，迭代的实现值的更新都在select语句后

…

select…from [table\_after\_constraint] 是这么工作的: 按顺序逐条阅读表的纪录。假设现在是第一条，select就从第一条中选取所需的东西；然后读取第二条，select从第二条中选取所需要的东西；如此继续。我们在这里用到用户定义的变量，其特色是在原基础表格的基础上追加一到多列，起到标记的作用，在计算机读一遍完成的时候形成一个可用的基础表格，然后再从这个表格上选择需要的纪录。这里标记的变化是通过上面给出的语句中from前的select 中给出的。我们具体来看一下，一开始，计算对tmp1和tmp2做join，即在tmp1后追加了3列，且他们的值都是初始值。分别为null，0，1。现在开始用select来选出第一条纪录。从而新的基础表的第一条记录为tmp1的第一条和由null，0，1生成的几个值。这是我们想象（因为select后的内容还没写出，所以说是想象）在这次选值的时候已经对@Department\_pre,@Salary\_pre,@count 重新赋值了。接下来select选出新的基础表格第二条纪录，这将包含tmp1的第二条纪录和由更新过的 @Department\_pre,@Salary\_pre,@count的值计算的来的select后面的内容。如此直到MySQL读完基础表格的最后一条纪录。

好了，现在我们来看看希望新的基础表格里有什么东西：他们要包含Department的名字信息，要包含Employee的名字信息，要包含Salary信息，然后要包含关于这个Salary在系中的排位信息。前三条从表timp1中选取，最后一条是我们要用前面的三个用户定义变量生成的。

…

select tmp1.Department, tmp1.Employee, tmp1.Salary

#下面应该对三个user-defined variable进行赋值且得出标记的值。注意是先得出标记

#的值，然后给三个用户变量赋值，至少这样不会错。

(Case

when tmp1.Department != @Department\_pre then @count:=1

when tmp1.Department = @Department\_pre and tmp1.Salary != @Salary\_pre, @count:=@count+1

else @count:=@count

) Record, #这里是得到标记的值，下面更新用户变量

@Department\_pre:=tmp1.Department,

@Salary\_pre:=tmp1.Salary

from (select D.Name as Department, E.Name as Employee, E.Salary as Salary

from Employee E, Department D

where E.DepartmentId = D.Id

order by E.DepartmentId, E.Salary desc, E.Name) tmp1,

(select @Department\_pre:=null, @Salary\_pre:=1, @count:=1 from dual) tmp2

现在可以完成最后的select:

select Department, Employee, Salary**, Record**

from (

select tmp1.Department, tmp1.Employee, tmp1.Salary,

(Case

when tmp1.Department != @Department\_pre then @count:=1

when (tmp1.Department = @Department\_pre and tmp1.Salary != @Salary\_pre) then @count:=@count+1

else @count:=@count

End) Record,

@Department\_pre:=tmp1.Department,

@Salary\_pre:=tmp1.Salary

from (select D.Name as Department, E.Name as Employee, E.Salary as Salary

from Employee E, Department D

where E.DepartmentId = D.Id

order by E.DepartmentId, E.Salary desc, E.Name) tmp1,

(select @Department\_pre:=null, @Salary\_pre:=0, @count:=1 from dual) tmp2

) Final

where Record < 4; #这里限制只要每个系前3的纪录

或者是

select Department, Employee, Salary

from (

select tmp1.Department, tmp1.Employee, tmp1.Salary,

If(tmp1.Department != @Department\_pre, @count:=1,@count:=@count),

If(tmp1.Department = @Department\_pre and tmp1.Salary != @Salary\_pre, @count:=@count+1,@count:=@count) Record,

@Department\_pre:=tmp1.Department,

@Salary\_pre:=tmp1.Salary

from (select D.Name as Department, E.Name as Employee, E.Salary as Salary

from Employee E, Department D

where E.DepartmentId = D.Id

order by E.DepartmentId, E.Salary desc) tmp1,

(select @Department\_pre:=null, @Salary\_pre:=1, @count:=1 from dual) tmp2

) final

where Record < 4; #这里限制只要每个系前3的纪录

类似的问题还有

top 1

top N 这里N是一个变量，查询可以通过创建函数实现

关于user-defined variable的例子可以参考一下网页

http://www.mysqldiary.com/user-defined-variables/

这里用到两个Syntax，第一个是case,有两种用法分别如下：

其一，Case

When condition1 Then Result1

[When condition1 Then Result2] …

[Else Resultn]

End

意思是如果condition1满足返回Result1的值（注，这里的result\_i可以为一个statement list）。否则检查condition2，如果为真则返回Result2的值，如果前都不真则返回else后面的值。如果没有else语句返回null。

其二，Case case\_value\_expression

When when\_value1 Then Result1

[When when\_value2 Then Result2]…

[Else Resultn]

End

意思是如果case后表达式的值在哪个when后出现就返回其后相应的resulti，否则返回else后的resultn。如果没有else返回null。

第二个语法是关于if的

语法是 If(expr1,expr2,expr3)

如果expr1为真则返回expr2的值，否则返回expr3的值。

Ifnull()的语法是

Ifnull(expr1,expr2)

如果expr1不是null则返回expr1的值，否则返回expr2的值。

**184. Department Highest Salary**

The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

+----+-------+--------+--------------+

| Id | Name | Salary | DepartmentId |

+----+-------+--------+--------------+

| 1 | Joe | 70000 | 1 |

| 2 | Henry | 80000 | 2 |

| 3 | Sam | 60000 | 2 |

| 4 | Max | 90000 | 1 |

+----+-------+--------+--------------+

The Department table holds all departments of the company.

+----+----------+

| Id | Name |

+----+----------+

| 1 | IT |

| 2 | Sales |

+----+----------+

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, Max has the highest salary in the IT department and Henry has the highest salary in the Sales department.

+------------+----------+--------+

| Department | Employee | Salary |

+------------+----------+--------+

| IT | Max | 90000 |

| Sales | Henry | 80000 |

+------------+----------+--------+

解答完上一道题，这道题就不是那么难了。至少这是上一道题的一种特殊情形。这里我们仅仅给出code。

select D.Name as Department, E.Name as Employee, T.Salary

from Employee E

inner join

(select DepartmentId,Max(Salary) as Salary

from Employee

group by DepartmentId) T

on

(T.DepartmentId=E.DepartmentId and T.Salary=E.Salary)

inner join

Department D

On D.Id=T.DepartmentId;

或者是

select D2.Name Department, E2.Name Employee, S.Salary

from (

select MAX(E.Salary) Salary, E.DepartmentId

from Employee E, Department D

where E.DepartmentId=D.Id

group by E.DepartmentId

) S,

Employee E2, Department D2

where S.Salary=E2.Salary

and E2.DepartmentId=D2.Id

and E2.DepartmentId=S.DepartmentId;

**183. Customers Who Never Order**

Suppose that a website contains two tables, the Customers table and the Orders table. Write a SQL query to find all customers who never order anything.

Table: Customers.

+----+-------+

| Id | Name |

+----+-------+

| 1 | Joe |

| 2 | Henry |

| 3 | Sam |

| 4 | Max |

+----+-------+

Table: Orders.

+----+------------+

| Id | CustomerId |

+----+------------+

| 1 | 3 |

| 2 | 1 |

+----+------------+

Using the above tables as example, return the following:

+-----------+

| Customers |

+-----------+

| Henry |

| Max |

+-----------+

首先创建表格

create table Customers (

Id tinyint primary key,

Name varchar(20));

create table Orders (

Id tinyint primary key,

CustomerId tinyint);

insert into Customers values

(1,’Joe’),

(2,’Henry’),

(3,’Sam’),

(4,’Max’);

insert into Orders values

(1,3),

(2,1);

解题思路：

要找的是没有点菜的顾客的名字。这涉及两个表格，名字从Customers表格来，用Orders表格进行选择。

方法一用left join或者是right join

基础表格这么构造出来：用Customers left join Orders，连接的条件是Customers.Id=Orders.CustomerId,如果没有满足条件，left join会在相应的customer纪录后添加null值。而这正是我们选择的依据。如果left join后的表格的某个纪录有null值，则选出。

select …

from Customers C left join Orders O

On C.Id=O.CustomerId

现在处理select后面的语句,以及限制的where语句

select C.Name as Customers

from Customers C left join Orders O

On C.Id=O.CustomerId

where O.Id is null;

方法二用子查询的方法实现：

基础表格是Customers，子查询在where里实现

select Name as Customers

from Customers

where Customers.Id not in (select CustomerId from Orders);

**182. Duplicate Emails**

Write a SQL query to find all duplicate emails in a table named Person.

+----+---------+

| Id | Email |

+----+---------+

| 1 | a@b.com |

| 2 | c@d.com |

| 3 | a@b.com |

+----+---------+

For example, your query should return the following for the above table:

+---------+

| Email |

+---------+

| a@b.com |

+---------+

首先创建表格。

create table Person (

Id tinyint,

Email varchar(30));

insert into Person values

(1,’a@b.com’),

(2,’c@d.com’),

(3,’a@b.com’);

解题思路：

还是可以用两种方法来解答。join 与子查询。实际上有多种方法。

第一种方法，源自让基础表格的每一条纪录中含有可以用来参考比较的字段。以此题为例，

一条记录中必须含有Id（用于挑选的），Email（用于挑选的），Id（用于比较的，存储相同的Email情况下最大或最小的id），Email（相应的比较的Email，用于建立表间的连接）。Code 如下

select …

from Person P1, (select max(tmp.Id) as Id, tmp.Email

from Person tmp group by tmp.Email) P2

where P1.Email = P2.Email

and …

现在完成select和where后面的部分：

select distinct(P1.Email)

from Person P1, (select max(tmp.Id) as Id, tmp.Email

from Person tmp group by Email) P2

where P1.Email = P2.Email

and P1.Id != P2.Id;

第二一类思路是将限制在子查询中实现：

a）将上面的想法在子查询中实现

b）因为用到distinct，我们可以试试用group by语句

想法是做一个基础表格，中间只含有Email，这条Email出现的次数，这可以用group

by 加count(\*) 来实现。然后以出现次数大不大于1来做挑选标准。Code为

select Email

from (select Email, count(\*) as Times from Person group by Email) tmp

where tmp.Times>1;

刚才我考虑了是否可以用一条查询语句实现上面的讨论，在想到group by后不能用where（我想加如where count(\*)>1）后放弃了，实际上这是可以用having语句实现的。

select Email

from Person group by Email having count(\*)>1;

以下补充一些where，group by，having，order by， limit的用法。

在select查询中，where查询只能对MySQL保存在硬盘上的数据进行查询，而使用as定义的变量或者where查询返回的结果集是处于内存中的。想要对内存中的数据进行查询可以使用having。

看起来MySQL的工作原理是

1. 基础表格
2. 用where语句对基础表格的纪录进行筛选，存入内存
3. 再用group by语句对筛选结果分类，之后在内存中操作
4. 接着用having语句对每一类进行类内筛选
5. 然后用order by排序
6. 最后用limit限制输出数目
7. select逐纪录选出想要的字段或者是计算出以字段为变量的函数值

在上面的逻辑中，as定义的变量又是怎样，是在什么时候进入内存的呢？应该讲as定义的变量是对内存的表格进行操作。在where之后进入。

**181. Employees Earning More Than Their Managers**

The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

+----+-------+--------+-----------+

| Id | Name | Salary | ManagerId |

+----+-------+--------+-----------+

| 1 | Joe | 70000 | 3 |

| 2 | Henry | 80000 | 4 |

| 3 | Sam | 60000 | NULL |

| 4 | Max | 90000 | NULL |

+----+-------+--------+-----------+

Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

+----------+

| Employee |

+----------+

| Joe |

+----------+

创建表格Employee

create table Employee (

Id tinyint primary key,

Name varchar(20),

Salary mediumint,

ManagerId tinyint);

Insert into Employee values

(1,’Joe’,70000,3),

(2,’Henry’,80000,4),

(3,’Sam’,60000,NULL),

(4,’Max’,90000,NULL);

解题思路：

题目问的是找出比他们经理赚的多的雇员的名字。在表格Employee 的一条记录中只含有雇员的信息，经理的Id，不含经理的工资信息。所以肯定要进行二次查询或是做jion来形成基础表格。这是比较简单的：

select E1.Name as Employee

from Employee E1

where E1.ManagerId !=null

and E1.Salary > (select E2.Salary

from Employee E2

where E2.Id = E1.ManagerId)

这里是查询，并不是修改，所以不会出现前面某题中提到的问题。下面使用inner join查询：

select E1.Name as Employee

from Employee E1, Employee E2

where E1.ManagerId = E2.Id

and E1.Salary >E2.Salary;

**180. Consecutive Numbers**

Write a SQL query to find all numbers that appear at least three times consecutively.

+----+-----+

| Id | Num |

+----+-----+

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 2 |

| 5 | 1 |

| 6 | 2 |

| 7 | 2 |

+----+-----+

For example, given the above Logs table, 1 is the only number that appears consecutively for at least three times.

首先创建表：

create table Logs (

Id tinyint primary key,

Num smallint);

insert into Logs values

(1,1),

(2,1),

(3,1),

(4,2),

(5,1),

(6,2),

(7,2);

解题思路：

题目要找的是连续出现至少3次的数字。如果Logs表的一条纪录只有一个数字的信息，无法比较。一个简单想法就是3个Logs表做两次join。另外一个想法是，如果想只读Logs表一遍，则要纪录前面读的两条记录中的数字信息。这可以用user-defined variable来实现。

先看第一种想法：

select distinct(E1.Num)

from Logs L1, Logs L2, Logs L3

where L1.Num = L2.Num

and L2.Num = L3.Num;

再来看第二种想法的实现。我们给原始的Logs表增加一列Record用于纪录连续读下来时同一个数字出现的次数：从1开始，出现1次记1，出现n 次记n；如果数字发生变化了归1，重新开始。结果为Record等于3时的Num的信息。我们先来写最里面的from后的内容，也就是初始的状态

select …

from (

select [想要的内容＋由用户变量产生的Record＋用户变量的迭代]

from Logs, (select @count :=1,@Num\_pre:=null from dual) tmp1

) tmp2

这一步相当于把Logs表格后都增加了一列tmp.count（实际的名字是tmp.@count）且每一条纪录的值都为1。当然我们只需要用它来生成新的基础表格第一条纪录。即select里的内容：select里要含有Num的信息，含有@count的当前值和@Num\_pre上条纪录里的Num值生成的当前Record的信息，含有@count的迭代信息

select Num,

(If(@Num\_pre=Num,@count:=@count+1,1)) as Record,

@Num\_pre:=Num #我们不再对@count做什么了，因为在上条语句中已经完成了

from Logs, (select @count:=1,@Num\_pre:=null from dual) tmp

现在我们可以完成整个查询语句：

select distinct(Num) as ConsecutiveNums**, Record**

from (

select Num,

(if(@Num\_pre=Num,@count:=@count+1,@count:=1)) as Record,

@Num\_pre:=Num

from Logs, (select @count:=1,@Num\_pre:=null from dual) tmp1

) tmp2

where Record=3;

**178. Rank Scores**

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.

+----+-------+

| Id | Score |

+----+-------+

| 1 | 3.50 |

| 2 | 3.65 |

| 3 | 4.00 |

| 4 | 3.85 |

| 5 | 4.00 |

| 6 | 3.65 |

+----+-------+

For example, given the above Scores table, your query should generate the following report (order by highest score):

+-------+------+

| Score | Rank |

+-------+------+

| 4.00 | 1 |

| 4.00 | 1 |

| 3.85 | 2 |

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

+-------+------+

这题只是上题第二种解法的一个应用。

首先创建表格

create table Scores (

Id tinyint primary key,

Score decimal(3,2));

insert into Scores values

(1,3.50),

(2,3.65),

(3,4.00),

(4,3.85),

(5,4.00),

(6,3.65);

解题思路：

要求给表格中的分数排序，使得相同的分数有相同的顺序，且不能有洞。从结果来看增加了一列，而且增加的列的值在随纪录变化，几乎可以肯定是要用user-defined variable。所以上面一道题的第二种解法放到这里很合适。先处理基础表格，第一个表是Scores，按照分数大小排序，要join的第二个表的作用是将user-defined variable初始化。如下：

select …

from (select Score from Scores order by Score desc) S1,

(select @Rank:=1 from dual) tmp

现在考虑select 后的部分。要达到两个目 的，第一，选出结果中要用到的变量；第二，处理好user-defined variable的迭代。结果要用的变量就是Score。Rank的变化，规则如下：假设现在是初始时刻，Rank的值直接从from里面来，为1，select后选出第一条纪录，Score为最大分数，Rank为1；计算机现在开始读基础表格的第二条纪录，执行select，选出Score，如果Score与前面的分数相同，Rank值要不变，如果值变了（实际是减少了），则Rank值加以。如此法则，读完基础表格为止。因此，Rank的变化可以用If语法来实现。

啊哈，我们还要用另外一个user-defined variable来存储上一条记录中的Score的值，用于读本条纪录时，其比较作用。而且，在设定Score\_pre的初始值时发现，设成null，Rank的初始值设成0会比较好，这样如果Score的值与Score\_pre的值一旦不同Rank的值即加1。这样第一步比较好处理。即：

select …

from (select Score from Scores order by Score desc) S1,

(select @Score\_pre:=null, @Rank:=0 from dual) tmp

select S1.Score, if(S1.Score=@Score\_pre,@Rank:=@Rank,@Rank:=@Rank+1) as Rank,@Score\_pre:=S1.Score

from (select Score from Scores order by Score desc) S1,

(select @Score\_pre:=null, @Rank:=0 from dual) tmp;

select Score, Rank

from ( select S1.Score, if(S1.Score=@Score\_pre,@Rank:=@Rank,@Rank:=@Rank+1) as Rank,@Score\_pre:=S1.Score

from (select Score from Scores order by Score desc) S1,

(select @Score\_pre:=null, @Rank:=0 from dual) tmp

) tmp2;

**177. Nth Highest Salary**

Write a SQL query to get the *n*th highest salary from the Employee table.

+----+--------+

| Id | Salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

For example, given the above Employee table, the *n*th highest salary where *n* = 2 is 200. If there is no *n*th highest salary, then the query should return null.

题目要求用create function来求解

create function getNthHighestSalary(N int) returns int

begin

return (

# Write your MySQL query statement below.

);

end

**176. Second Highest Salary**

Write a SQL query to get the second highest salary from the Employee table.

+----+--------+

| Id | Salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

For example, given the above Employee table, the second highest salary is 200. If there is no second highest salary, then the query should return null.

首先创建表格

create table Employee (

Id tinyint primary key,

Salary smallint);

解题思路：

**175. Combine Two Tables**

Table: Person

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| PersonId | int |

| FirstName | varchar |

| LastName | varchar |

+-------------+---------+

PersonId is the primary key column for this table.

Table: Address

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| AddressId | int |

| PersonId | int |

| City | varchar |

| State | varchar |

+-------------+---------+

AddressId is the primary key column for this table.

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:

FirstName, LastName, City, State

首先创建表格

create table Person (

PersonId int primary key,

Firstname varchar(20),

LastName varchar(20));

create table Address (

AddressId int primary key,

PersonId int,

City varchar(20),

State varchar(20));

解答：这一题是基本的left join。Code如下：

select P.FirstName, P.LastName, A.City, A.State

from Person P left join Address A

on P.PersonId = A.PersonId;